Wprowadzenie do Algorytmiki. Preprocesor. Biblioteka Standatdowa. Operatory bitowe. Branch & Bound.

Artur Laskowski

25 listopada 2021, Poznań

Etapy kompilacji w C++

- Wejście: Plik źródłowy (*.cpp)
- **Preprocespr**: Plik źródłowy (*.cpp)
- Kompilator: Plik obiektowy (*.o)
- Konsolidator (linker): Plik wykonywalny (*.exe/*.out)

Preprocesor

- Przygotowuje plik źródłowy do kompilacji
- Dołącza wszystkie wymagane nagłówki (pliki *.h)
- Rozwija makra
- Sterują nim instrukcje rozpoczynające się od #

Przykład

```
#define MAX 400

int main() {
    int n = MAX;
    return 0;
}
```

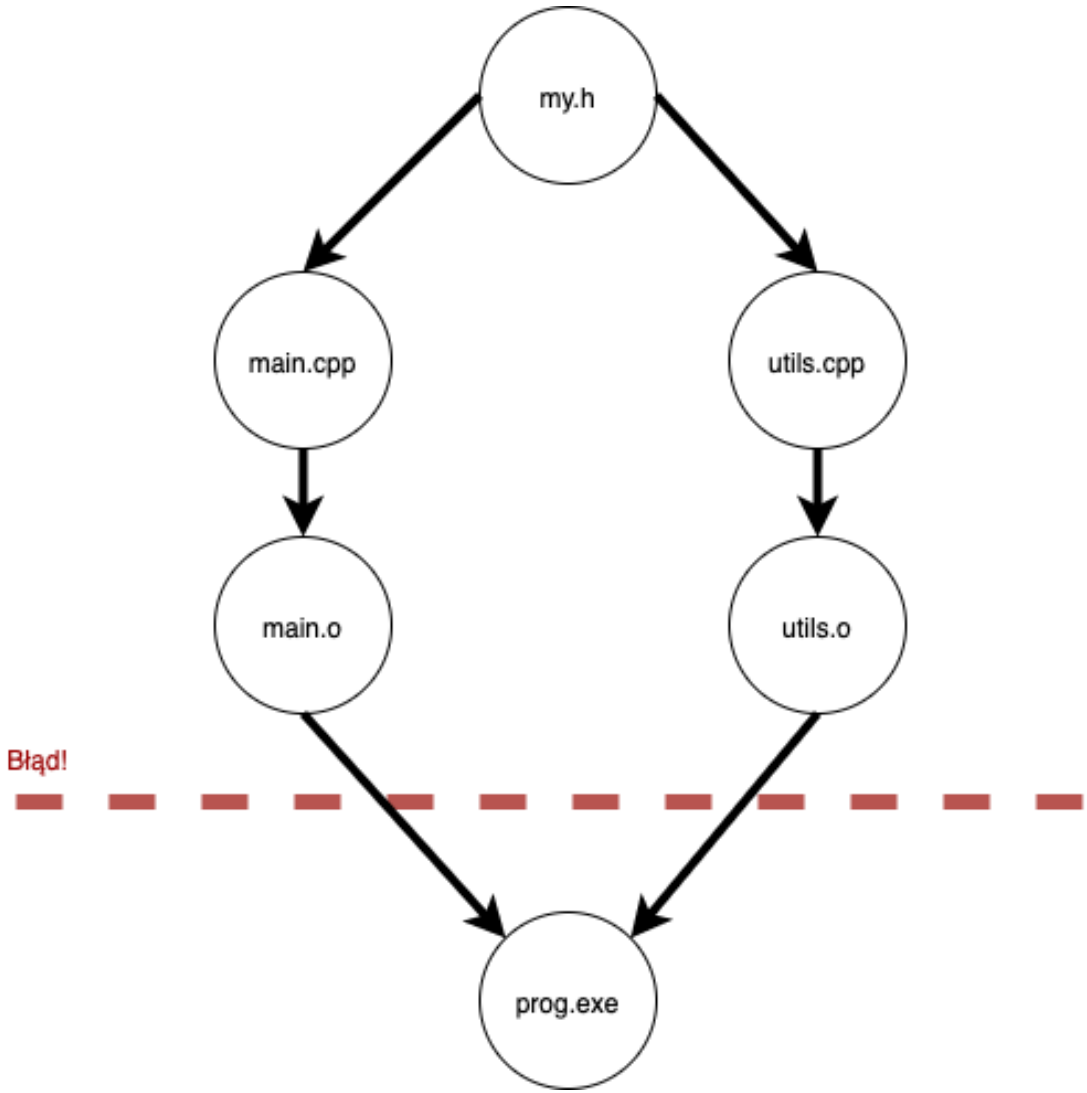
```
int main() {
    int n = 400;
    return 0;
}
```

Przykład 2

```
/*  my.h    */

int Foo = 400;

/*  my.cpp  */

#include "my.h"

int main() {
    int n = Foo;
    return 0;
}
```

```
int Foo = 400;

int main() {
    int n = Foo;
    return 0;
}
```

Łączenie plików



Makra w C++

- Rozpoczynają sie od instrukcji define
- Definiują tekst, który będzie przez procesor zamieniony na inny
- Mogą zawierać argumenty
- Kolejne linie oddzielone \
- Są wydajniejsze od funkcji
- Generują dłuższy kod

Przykłady

```
#define MAX 400
#define HELLO "Hello World"
#define MYF very_long_function_name
#define SQR(a) \
    (a)*(a)
```

Uwaga na nawiasy

```
// Błędna definicja:

#define SQRT(a) a*a

// Uzycie:
SQRT(3+4)

// Wygeneruje kod:
3+4*3+4
```

Brak sprawdzania typów

```
// Kod:

#define MAX(a, b) ((a < b) ? b : a)
cout << MAX("20", "10") << endl;

// Wypisze:
10  // Jest to łańcuch o większym wskaźniku
```

Uzyteczne makra

```
#define FOR(I,N) for(int I=0;I<(N);I++)
#define FORD(I,N) for(int I=N-1;I>=0;I--)
#define REP(I,A,B) for(int I=(A);I<=(B);I++)
#define REPD(I,A,B) for(int I=(A);I>=(B);I--)
#define VAR(V,init) __typeof(init) V=(init)
#define FOREACH(I,C) \
    for(VAR(I,(C).begin());I!=(C).end();I++)
#define ALL(X) (X).begin(),(X).end()
#define PB push_back
#define MP make_pair
#define FI first
#define SE second
```

Podsumowanie

- Czasami zwiększają czytelność
- Przyspieszają pisanie kodu
- Niezależne od kompilatora
- Ułatwiają popełnianie błędów
- Wydłużają kod wynikowy

Typedef ;)

```
typedef ULL unsigned long long;

typedef VI vector <int>;
```

Znajdowanie max

```
int max(int a, int b) {
    return (a < b) ? b : a;
}

double max(double a, double b) {
    return (a < b) ? b : a;
}

float max(float a, float b) {
    return (a < b) ? b : a;
}
```

Szablony

```
template<class T>
T max(T a, T b) {
    return (a < b) ? b : a;
}
```

Biblioteka standardowa

- Dostępna zawsze wraz z kompilatorem
- Znajdują się w niej implementacje:
  - struktur danych
  - algorytmów
- Łatwa i szybka w użyciu

Operacje wejścia/wyjścia

```cpp
#include <iostream>
using namespace std;

int x;
float f;

// ...

cin >> x >> f;
cout << x * f;
```

Nagłówki w STL

Nie dajemy rozszerzenia .h przy załączaniu.
Dla standardowych bibliotek z C dajemy "c" na początku:

- cstdio, zamiast stdio.h
- cstdlib
- cmath
- cctype
- ...

Para

```cpp
#include <utility>

pair <int, double> p;

p = make_pair(n, 5.0);

cout << p.first * p.second << endl;
```

Łańcuch znaków

```
#include <string>

string a = "abc";

a = string("abfff");
cin >> a;

a[a.length() - 1] = 'b';
cout << a;
```

Łańcuch znaków

```
a = a + a;
if(a < b) {
    // ...
}

strlen(a.c_str());
```

Nagłówek cctype

- islower(c)
- isupper(c)
- isdigit(c)
- isxdigit(c) - cyfra 16tkowa
- isalpha(c)
- isalnum(c)
- isspace(c) - biały znak

Strumienie

```cpp
#include <sstream>

ios_base::sync_with_stdio(false);

istringstream iss("3 4 5");
ostringstream oss;

iss >> a;
iss >> b >> c;
oss << a + b + c;
cout << oss.str();
```

Wektor

```cpp
#include <vector>

vector <int> v;

vector <int> w(15);

vector <int> q(15, 0);

v.resize(15);

w.reserve(100);
```

Wektor

```
v[5] = 56;

v.clear();

v.push_back(88);

v.pop_back();

v.front() = 5;
v.back() = 4;

cout << v.size() << endl;
```

Iterator

```cpp
for (int i = 0; i < v.size(); ++i) {
    cout << v[i] << endl;
}

for (vector<int>::iterator i = v.begin(); i != v.end(); ++i)
    cout << *i << endl;
}

for (auto i: v) {
    cout << *i << endl;
}
```

Kolejka

```cpp
#include <queue>

queue <int> q;

q.push(5);

while(!q.empty()) {
    cout << q.front() << endl;
    q.pop();
}
```

Kolejka dwukierunkowa

```
deque <int> dq;

dq.push_front(2);
dq.push_back(4);

cout << dq.front() + dq.back();

dq.pop_front();
dq.pop_back();
```

Kolejka priorytetowa

```
priority_queue <int> pq;

pq.push(5);

while(!.pq.empty()) {
    cout << pq.top();
    pq.pop();
}
```

Kolejka priorytetowa, priorytety

```
pair<double, string> array[100];

bool comparator(int idx_a, int idx_b) {
    if (array[idx_a].second > array[idx_b].second) {
        return true;
    } else if (array[idx_a].second < array[idx_b].second)) {
        return false;
    } else if (array[idx_a].first > array[idx_b].first) {
        return true;
    } else {
        return false;
    }
}

priority_queue <int, vector<double>, comparator> q;
```

Mapa

$$include < map >$$

- Iterator przegląda ją w kolejności posortowanych kluczy
- Iterator wskazuje na parę (klucz, wartość)

Zbiór

include < *set* >

- Umożliwiają szybkie sprawdzenie, czy dany element istnieje
- Łatwo wypisać elementy w kolejności posortowanej

Bitset

```
vector <bool> v;
bitset <100> v;
```

Algorytmy

include < algorithm >

```
int t[100];
vector <int> v(100);

sort(t, t+100);
sort(t, t+n);
sort(b.begin(), v.end());
sort(v.begin(), v.begin() + 100);
```

Algorytmy

- stable_sort - sortowanie stabilne
- unique - usuwa duplikaty w posortowanym ciągu
- reverse - odwraca kolejność elementów
- nth_element - n-ta statystyka pozycyjna
- lower_bound, upper_bound, binary_search - wyszukiwanie binarne
- min, max, swap
- set_union, _intersection, _difference

Algorytmy

```
vector <int> v;
sort(v.begin(), v.end());
v.erase(unique(v.begin(), v.end()), v.end());
```

Algorytmy

```
for(int i = 0; i < n; ++i) {
    perm[i] = i;
}
do {
    dowork(perm);
} while(next_permutation(perm, perm + n));
```

Największy wasz przyjaciel

http://www.cplusplus.com/
https://devdocs.io/cpp/
https://en.cppreference.com/w/

Zapis Binarny

Na *i*-tej pozycji zapisuj binarnego liczby występuje *i*-ta potęga liczby 2. Jeżeli dany bit jest ustawiony na 1 to taka potęga jest częścią składową zapisanej liczby, jeśli nie to nie.

Przykład:

$15 = 1111$

$19 = 10011$

Zapis Binarny w C++

W C++ liczbę można zapisać na 3 sposoby:

$int\ a\ =\ 15;$

$int\ a\ =\ 0b10011;$

$int\ a\ =\ 0x13;$

Zapis Binarny w C++

Dla liczb bez znaku zapisujmy z końcówką U.
Dla liczb dużych (większych od int) końcówka LL.

$$unsigned\ a\ =\ 12'413U;$$
$$long\ long\ a\ =\ 13'413'513'513LL;$$
$$unsigned\ long\ long\ a\ =\ 1'315'131'355ULL;$$

Operatory Logiczne w C++

Operują na wartościach *Prawda* oraz *Falsz*.
W C++ 0 to *Falsz*, a wszystkie inne wartości to *Prawda*.

&& - operator "*i*"

|| - operator "*lub*"

! - operator negacji

Przykład:

$$if\,(a-b)printf\,(\text{"}Rozne\text{"});$$

Operatory Logiczne w C++

Działają na poszczególnych bitach liczb.
Wykonują operację na wszystkich bitach liczby pokolei.

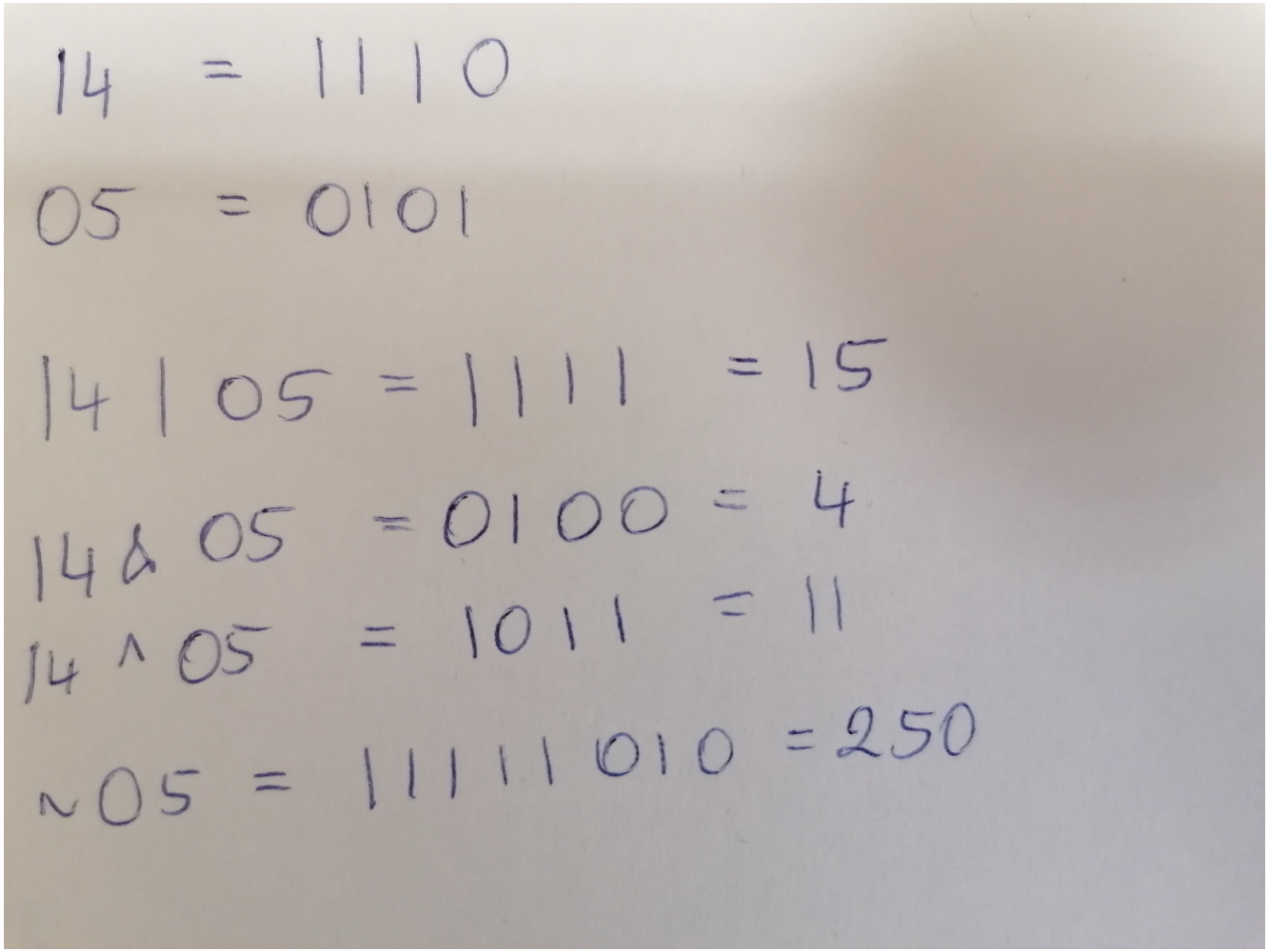
& - operator "*i*"

| - operator "*lub*"

$\sim$ - operator negacji

^ - operator "*albo*"

Przykład


14 = 1110
05 = 0101
14 | 05 = 1111 = 15
14 & 05 = 0100 = 4
14 ^ 05 = 1011 = 11
~05 = 11111010 = 250

Przesunięcia bitowe

$<<$

- Przesunięcie bitowe w lewo
- Dodajemy 0 z prawej strony
- $14 << 2 = 1110 << 2 = 111000 = 56$
- Operacja jest równoważna do mnożenia przez 2

$>>$

- Przesunięcie bitowe w prawo
- Obcina bity z prawej strony
- $14 >> 2 = 1110 >> 2 = 11 = 3$
- Operacja jest równoważna do dzielenia bez reszty przez 2

Przydatne operacje

```
int x = (v & (1 << k)) != 0;
```

```
int y = (v |= 1 << k);
```

```
int z = (v &= ~(1 << k));
```

Przydatne operacje

```
int x = (v & (1 << k)) != 0;
```

Sprawdzamy k-ty bit

```
int y = (v |= 1 << k);
```

Ustawiamy k-ty bit

```
int z = (v &= ~(1 << k));
```

Usuwamy k-ty bit

Zliczanie zapalonych bitów

```
int bity = 0;
for(int i = 0; i < 32; ++i) {
    if(v & (1 << i)) ++bity;
}
```

Zliczanie zapalonych bitów

```
int bity2 = __builtin_popcount(v);
```

Generowanie podzbiorów

- Dany jest zbiór n elementowy
- Wszystkie jego podzbiory mogą być reprezentowane przez liczby $0..2^n - 1$
- Bit zapalony oznacza element występujący w podzbiorze

Zadanie QAP

- Dane jest n fabryk i n odbiorców
- Dana jest macierz odległości pomiędzy fabrykami i odbiorcami
- Każdą fabrykę należy przyporządkować do innego odbiorcy, tak aby suma odległości była minimalna

Zadanie QAP

	1	2	3	4	5
A	5	3	7	4	9
B	8	6	3	5	4
C	7	6	2	4	7
D	5	6	5	7	3
E	4	3	8	5	6

Laboratoria

https://www.hackerrank.com/wda-05-2021